Lectures 13, 14
Scribe: Anupam
Last revised

# Lectures 13, 14

## 1 Streaming Algorithms

The streaming model is one way to model the problem of analyzing massive data. The model assumes that the data is presented as a stream $(x_1, x_2, \cdots, x_m)$, where the items $x_i$ are drawn drawn from a universe of size $n$. Realtime data like server logs, user clicks and search queries are modeled by streams. The available memory is much less than the size of the stream, so a streaming algorithm must process a stream in a single pass using sublinear space.

We consider the problem of estimating stream statistics using $O(\log^c n)$ space. The number of occurrences of element $i$ in the stream is denoted by $m_i$. The frequency moments $F_k = \sum_i m_i^k$ are natural statistics for streams.

The moment $F_0$ counts the number of distinct items, an algorithm that estimates $F_0$ can be used to find number of unique visitors to a website, by processing the stream of ip addresses. The moment $F_1$ is trivial as it is the length of the stream while computing $F_2$ is more involved. The streaming algorithms for estimating $F_0$ and $F_2$ rely on pairwise independent hash functions, which we introduce next.

### 1.1 Counting distinct items

Exactly counting the number of distinct elements in a stream requires $O(n)$ space, we will present a randomized algorithm that estimates the number of distinct elements to to a multiplicative factor of $(1 \pm \epsilon)$ with high probability using poly$(\log n, \frac{1}{\epsilon})$ space. The probabilities are over the internal randomness used by the algorithm, the input stream is deterministic and fixed in advance.

#### 1.1.1 Exact counting requires $O(n)$ space

Suppose $A$ is an algorithm that counts the number of distinct elements in a stream $S$ with elements drawn from $[n]$. After executing $A$ on the input stream $S$ it acts as a membership tester for $S$. On input $x \in [n]$ the count of distinct items increases by 1 if $x \notin S$ and stays the same if $x \in S$. The internal state of $A$ must contain enough information to distinguish between the $2^n$ possible subsets of $[n]$ that could have occurred in $S$. The algorithm requires $O(n)$ bits of storage to distinguish between $2^n$ possibilities.

#### 1.1.2 A toy problem

Consider the following simpler version of approximate counting: The output should be 'yes' if the number of distinct items $N$ is more than $2k$, 'no' if $N$ is less than $k$ and we do not care about the output if $k \leq N \leq 2k$.

1. Choose a uniformly random hash function $h : [n] \to [B]$, where the number of buckets $B = O(k)$.
2. Output 'yes' if there is some $x_i \in S$ such that $h(x_i) = 0$ else output 'no'.

The value $h(x)$ is uniformly distributed on $[B]$, so for all $x \in U$ we have $\Pr_{h \in \mathcal{H}}[h(x) = 0] = 1/B$. If there are at most $k$ distinct items in the stream, the probability that none of the $N$ items hash to 0 is,

$$\Pr[A(x) = No \mid N \leq k] = \left(1 - \frac{1}{B}\right)^N \geq \left(1 - \frac{1}{B}\right)^k$$

If the number of elements is greater than $2k$ then the probability that the algorithm outputs no is,

$$\Pr[A(x) = No \mid N > 2k] = \left(1 - \frac{1}{B}\right)^N \leq \left(1 - \frac{1}{B}\right)^{2k}$$

The gap between the probability of the output being 'no' for the two cases is a constant for $B = O(k)$.

However, specifying a random hash function requires $O(n \log B)$ bits of storage, the truth table must be stored to evaluate the hash function. The memory requirement can be reduced by choosing $h$ from a hash function family $\mathcal{H}$ of small size having good independence properties.

**2-wise independent hash functions:** The property required from $\mathcal{H}$ is 2-wise independence, informally a hash function family is 2 wise independent if the hash value $h(x)$ provides no information about $h(y)$.

Claim 1
*The family $\mathcal{H} : [n] \to [p]$ consisting of functions $h_{a,b}(x) = ax + b \mod p$ where $p$ is a prime number greater than $n$ and $a, b \in \mathbb{Z}_p$ is 2-wise independent,*

$$\Pr_{a,b}[h(x) = c \wedge h(y) = d] = \frac{1}{p^2} \qquad \forall x \neq y$$

Proof: If $h(x) = c$ and $h(y) = d$ then the following linear equations are satisfied over $\mathbb{Z}_p$,

$$ax + b = c \qquad ay + b = d$$

The linear system has a unique solution $(a, b)$ as the determinant $(x - y) \neq 0$ for distinct $x, y$. The claim follows as $|H| = p^2$ and there is a unique function such that $h(x) = c$ and $h(y) = d$.
□

This construction of 2 wise independent hash function families generalizes to $k$ wise independent families by choosing degree $k$ polynomials. For the streaming algorithm we require a 2-wise independent hash function family $\mathcal{H} : [n] \to [B]$ where $B$ is not a prime number, the family $h_{a,b} = (ax+b \mod p) \mod B$ for a prime larger than $p$ is approximately 2 wise independent.

## 1.2 Analysis

We analyze the algorithm using a random hash function from a pairwise independent family $\mathcal{H} : [n] \rightarrow [4k]$. From claim 1, it follows that $\Pr_{a,b}[h(x) = 0] = 1/B$ for all $x \in [U]$. If there are $k$ elements in the stream the probability of some element being hashed to 0 can be bounded using the union bound $\Pr[\cup A_i] \leq \sum \Pr[A_i]$,

$$\Pr[A(x) = Yes \mid N < k] \leq \frac{k}{B} = \frac{1}{4} \tag{1}$$

The inclusion exclusion principle is used to show that the probability of the output being yes is large if there are more than $2k$ elements in the stream. Truncating the inclusion exclusion formula to the first two terms yields $\Pr[\cup A_i] \geq \sum \Pr[A_i] - \sum \Pr[A_i \cap A_j]$. Using pairwise independence,

$$\Pr[A(x) = Yes \mid N \geq 2k] \geq \frac{2k}{B} - \frac{2k.(2k-1)}{2B} \geq \frac{2k}{B}(1 - \frac{k}{B}) = \frac{3}{8} \tag{2}$$

The yes and no cases are separated by a gap of 1/8, the memory used by the algorithm is $O(\log n)$ as numbers $a, b$ need to be stored. Using a combination of standard tricks, the quality of approximation can be improved to $1 \pm \epsilon$.

## 1.3 A $1 \pm \epsilon$ approximation:

The probability of obtaining a correct answer is boosted to $1 - \delta$ by running the algorithm with several independent hash functions using the following simplified version of Chernoff bounds,

Claim 2
*If a coin with bias $b$ is flipped $k = O(\frac{\log(1/\delta)}{\epsilon^2})$ times, with probability $1 - \delta$ the number of heads $\widehat{b}$ satisfies $bk(1 - \epsilon) \leq \widehat{b} \leq bk(1 + \epsilon)$.*

The algorithm is run for $O(\log 1/\delta)$ independent iterations and the output is 'yes' if the fraction of yes answers is more than 5/16. Applying the claim for the yes and no cases, it follows that the correct answer is obtained with probability at least $1 - \delta$.

The number of distinct items $N$ can be approximated to a factor of 2 using the binary search trick. The algorithm is run simultaneously for the $\log n$ intervals $[2^k, 2^{k+1}]$ for $k \in [\log n]$. If $N \in [2^k, 2^{k+1}]$ then with high probability the first $k-1$ runs answer 'yes', the answer for the $k$-th run is indeterminate and the last $\log n - k - 1$ runs answer 'no'. The first no in the sequence of answers occurs either for $[2^k, 2^{k+1}]$ or $[2^{k+1}, 2^{k+2}]$, the left end point of the interval where the transition occurs satisfies $\frac{N}{2} \leq L \leq 2N$.

The third trick is to replace 2 by $1 + \epsilon$ in equations (**??**), (**??**) and change parameters appropriately in the boosting part to approximate the number of distinct items in the stream up to a factor of $1 \pm \epsilon$.

The space requirement of the algorithm is $O(\log n . \log_{1+\epsilon} n . \frac{\log(1/\delta)}{\epsilon^2})$, the $\log n$ is the amount of memory required to store a single hash function, the $\log_{1+\epsilon} n$ is the number of intervals considered and $\frac{\log(1/\delta)}{\epsilon^2}$ is the number of independent hash functions used for each interval.

## 2 Estimating $F_2$

The hash function $h$ is chosen from a 4-wise independent family $\mathcal{H} : [n] \rightarrow \pm 1$. The algorithm outputs $Z = (\sum_i h(x_i))^2$ as an estimate for $\mu$, the memory requirement is $O(\log n)$. The analysis will show that $E[Z^2] = F_2$ and that the variance is small. Denoting the hash value $h(j)$ by $Y_j$ we have,

$$Z = \sum_{i \in [m]} h(x_i) = \sum_{j \in S} Y_j m_j$$

The expectation of $Z^2$ can be computed by squaring and using the 2 wise independence of the hash function to cancel out the cross terms,

$$E[Z^2] = \sum_j E[Y_j^2] m_j^2 + \sum_{i,j} E[Y_i] E[Y_j] m_i m_j = \sum_i m_i^2 = F_2$$

A variance calculation is required to ensure that we obtain the correct answer with sufficiently high probability. Recall that the variance of a random variable $X$ is equal to $E[X^2] - E[X]^2$, the variance calculation requires computing the fourth moment of $Z$,

$$E[Z^4] = \sum_i E[Y_i^4 m_i^4] + 6 \sum_{i,j} E[Y_i^2 Y_j^2 m_i^2 m_j^2] = \sum_i m_i^4 + 6 \sum_{i,j} m_i^2 m_j^2$$

The variance of $Z^2$ can now be computed,

$$Var(Z^2) = E[Z^4] - E[Z^2]^2 = 4 \sum m_i^2 m_j^2 < 2F_2^2$$

The Chebyshev inequality is useful for bounding the deviation of a random variable from its mean,

$$\Pr[|X - \mu| \geq \epsilon F_2] \leq \frac{Var(X)}{\epsilon^2 F_2^2}$$

The variance is too large for Chebyshev's inequality to be useful. The variance can be reduced by running the procedure over $k = 2/\delta\epsilon^2$ independent iterations, with the output being $Z = \frac{1}{k} \sum_{i \in [k]} Z_i^2$.

The expectation $E[Z] = \mu$ by linearity and the the variance can be calculated using relations $Var[cX] = c^2 Var[X]$ and $Var(X + Y) = Var(X) + Var(Y)$ for independent random variables $X$ and $Y$.

$$Var[Z] = \sum_{i \in [k]} Var\left[\frac{Z_i^2}{k}\right] \leq \frac{2F_2^2}{k}$$

Applying the Chebychev inequality for $Z = \frac{1}{k} \sum_{i \in [k]} Z_i^2$ with $k = \frac{2}{\delta\epsilon^2}$yields $\Pr[|Z - \mu| \geq \epsilon F_2] \leq \delta$. The output of the algorithm $Z$ is therefore a $(1 \pm \epsilon)$ approximation for $\mu$ with probability at least $1 - \delta$. The memory requirement for the algorithm is $O(\log n/\epsilon^2)$.